

Manuelle Korrektur

JavaDoc

Was muss alles vorhanden sein?

Eine Dokumentation muss folgende Informationen enthalten:

- Kurze Beschreibung:
 - Jede Klasse, Methode oder Attribut sollte eine prägnante Beschreibung haben, die den Zweck und die Verwendung klar vermittelt.
 - Verwende vollständige Sätze, um die Lesbarkeit zu verbessern.
- `param`-Tags:
 - Für Methoden mit Parametern sollten Parameter-Tags vorhanden sein, um die Bedeutung jedes Parameters zu erklären.
 - Jeder Parameter sollte einzeln dokumentiert werden.
- `return`-Tag:
 - Wenn eine Methode einen Rückgabewert hat, sollte ein return-Tag vorhanden sein, um den Rückgabetyt und eine kurze Beschreibung des Rückgabewerts anzugeben.
- `throws`-Tag:
 - Falls eine Methode Checked-Exceptions auslösen kann, sollte ein throws-Tag vorhanden sein, um die möglichen Ausnahmen zu dokumentieren

Ablauf

1. Navigiere in Moodle zum Abschnitt "Intern" und wähle den Reiter "Manuelle Korrektur" aus.
2. Suche den Ordner "Dokumentation" im Zusammenhang mit der Hausübung "Hxy".
3. Lade die entsprechende Datei herunter.
4. Die Tabelle ist folgendermaßen aufgebaut:
 1. `id` : Eindeutige Kennung
 2. `id_tu` : TU-ID der zu korrigierenden Person
 3. `name_expected` : Erwarteter Name der Klasse, Methode oder des Attributs
 4. `name_actual` : Aktueller Name der Klasse, Methode oder des Attributs
 5. `valid` : Bewertung der Dokumentation (n = nicht ausreichend, y = ausreichend)
 6. `deviation` : Gibt an, ob es eine Abweichung gibt oder eine Diskrepanz besteht
 7. `documentation` : Zu korrigierende JavaDoc
5. Überprüfe die Richtigkeit der Dokumentation. Die Spalten `valid` und `documentation` sind für euch relevant.
6. Lade die korrigierte Datei in Moodle in "Hochladen der bewerteten Dokumentationen" hoch.

Überprüfung

- Plausible Dokumentation:

- Überprüfe, ob die Dokumentation für jede Klasse, Methode oder Attribut plausibel ist und den Zweck sowie die Verwendung klar vermittelt.
- Achte darauf, dass die Beschreibungen vollständig und verständlich sind.
- Man soll in kurzen Worten erklären können, wofür ein Konstrukt da ist, ohne dabei den Code gesehen zu haben, um es zu verstehen.
- Punktzahl nach eigenem Ermessen:
 - Trage die Punktzahl nach eigenem Ermessen ein, solange die Dokumentation für die jeweilige Klasse, Methode oder Attribut plausibel erscheint.
 - Verwende den valid-Eintrag, um die Bewertung (0 = nicht ausreichend, 1 = ausreichend) festzuhalten.
- Getter-Methoden:
 - Bei Getter-Methoden könnt ihr eventuell nachsichtig sein, wenn die Beschreibung fehlt, solange der `return`-Tag vollständig und korrekt ist.
 - Achte darauf, dass der Rückgabetyt und die Beschreibung des zurückgegebenen Werts im `return`-Tag klar sind.
- Parameter-Tag:
 - Falls die Parameter-Tag nicht dokumentiert sind, könnt ihr eventuell nachsichtig sein, wenn die Beschreibung die Parameter erläutern.
- Sprache der Dokumentation:
 - Die Sprache der Dokumentation ist entweder deutsch oder englisch. Alle anderen Sprachen werden als nicht ausreichend bewertet.

Beispiele

Klasse

Positiv-Beispiel

```
/**
 * Represents a 2D geometric point with x and y coordinates.
 */
public class Point {

}
```

Negativ-Beispiel

- Beschreibung nicht ausreichend

```
/**
 * A point.
 */
public class Point {

}
```

Interface

Positiv-Beispiel

```

/**
 * Defines the contract for objects that can be compared.
 */
public interface Comparable {

    /**
     * Compares this object with another object for order.
     *
     * @param other the object to be compared
     * @return a negative integer, zero, or a positive integer as this object is less than, equal to, or greater than
     the specified object
     */
    int compareTo(Object other);
}

```

Negativ-Beispiel

- Beschreibung nicht ausreichend

```

/**
 * Comparison interface.
 */
public interface Comparable {

    int compareTo(Object other);
}

```

Enum

Positiv-Beispiel

```

/**
 * Represents the days of the week.
 */
public enum Day {

    MONDAY,
    TUESDAY,
    WEDNESDAY,
    THURSDAY,
    FRIDAY,

```

```
SATURDAY,  
SUNDAY;  
}
```

Negativ-Beispiel

- Beschreibung nicht ausreichend

```
/**  
 * Days enum.  
 */  
public enum Day {  
  
    MONDAY,  
    TUESDAY,  
    WEDNESDAY,  
    THURSDAY,  
    FRIDAY,  
    SATURDAY,  
    SUNDAY;  
}
```

Konstruktor

Positiv-Beispiel

```
/**  
 * Creates a new point with the specified coordinates.  
 *  
 * @param x the x-coordinate of the point  
 * @param y the y-coordinate of the point  
 */  
public Point(double x, double y) {  
    this.x = x;  
    this.y = y;  
}
```

Negativ-Beispiel

- Beschreibung nicht ausreichend

```

/**
 * Creates a point.
 *
 * @param x the x-coordinate of the point
 * @param y the y-coordinate of the point
 */
public Point(double x, double y) {
    this.x = x;
    this.y = y;
}

```

Methode

Positiv-Beispiel

- Sofern `return`-Tag vollständig ist, kann man aus Kulanz bei fehlender Beschreibung die Bewertung durchgehen lassen

```

/**
 * Retrieves the x-coordinate of this point.
 *
 * @return the x-coordinate of this point
 */
public double getX() {
    return x;
}

```

```

/**
 * Computes the magnitude (length) of the vector represented by this instance.
 *
 * @return magnitude of the vector
 */
public double magnitude() {
    return Math.sqrt(x * x + y * y);
}

```

Negativ-Beispiel

- Fehlendes `return`-Tag

```

/**
 * The x-coordinate of this point.

```

```
*/  
public double getX() {  
    return x;  
}
```

- Die Dokumentation sollte nicht auf die Implementierung eingehen. Man soll anhand der Dokumentation in kurzen Worten verstehen können, was die Funktionalität der Methode ist, ohne explizit zu erwähnen, wie das Ganze umgesetzt wurde.

```
/**  
 * Computes the magnitude (length) of the vector using the Math.sqrt method with the input  $x * x + y * y$ .  
 *  
 * @return magnitude of the vector  
 */  
public double magnitude() {  
    return Math.sqrt(x * x + y * y);  
}
```

Revision #1

Created 8 October 2024 13:52:10 by Svana Esche

Updated 8 October 2024 13:52:10 by Svana Esche